Embedded Systems 1

Lab Report 1

Gavin McKim

2/28/2019

Purpose:

The purpose of this lab is to get used to utilizing and implementing vhdl code. The lab introduces the basics of writing source code and testing it with testbench code. It does this while also introducing the basic building blocks of many digital circuits. Specifically, the lab introduces the counter and walks us through the many uses of it. It walks us through the many creative ways to use a counter to solve problems.

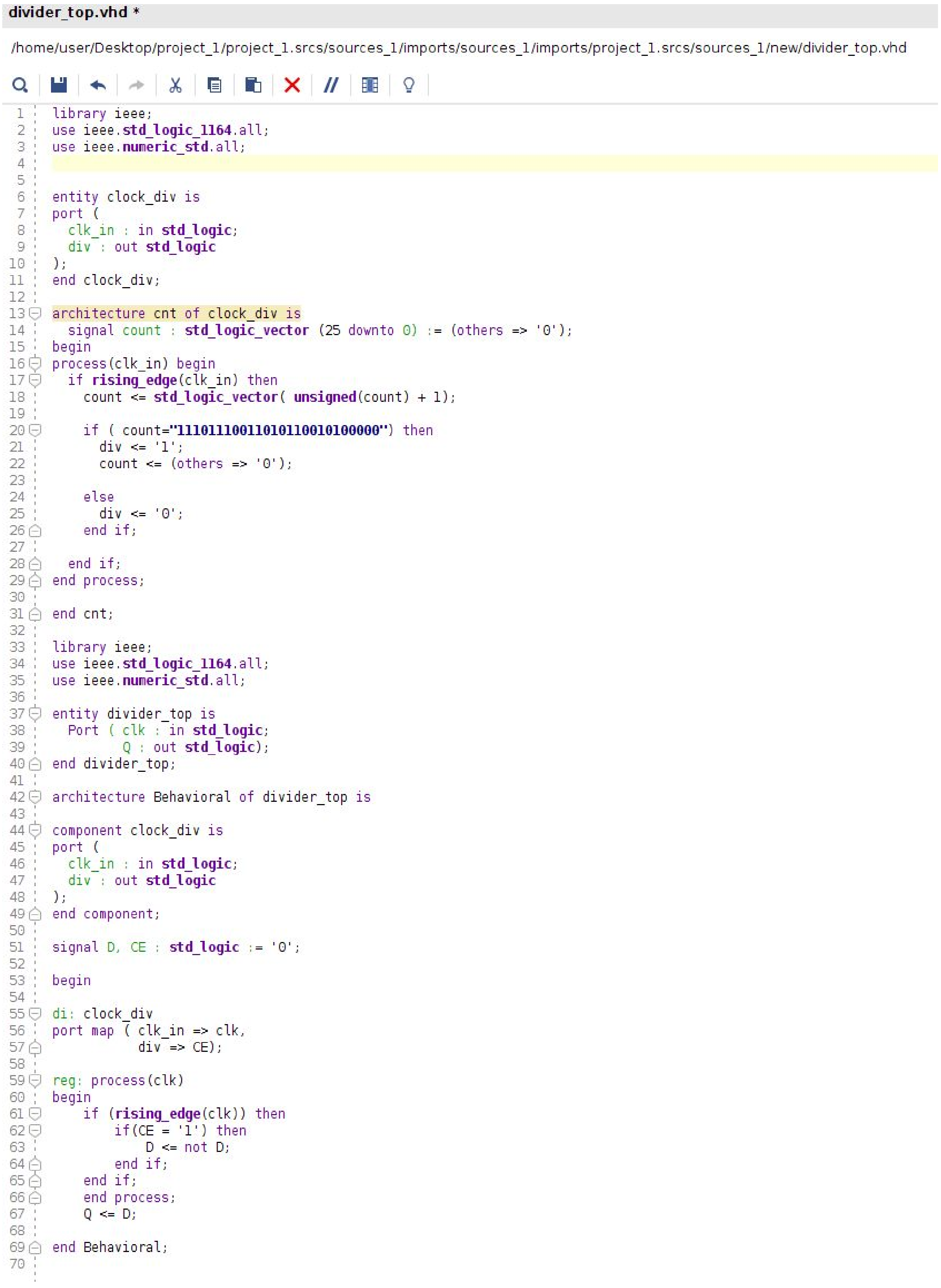
1. My Clock is Just Right

Theory

In this first part, the circuit should be able to take an input signal with a frequency of 125MHz, and step it down into a signal with a frequency of 2Hz. It does this using a counter. The counter increments up on every rising edge of the 125MHz clock. When the counter reaches 62.5M, the circuit outputs a pulse and resets the counter. 62.5M is used since it is the ratio between 125 MHz and 2 Hz. This design should output a pulse at a frequency of 2 Hz. In other words, it should output a pulse every 0.5 seconds.

Design

Code:



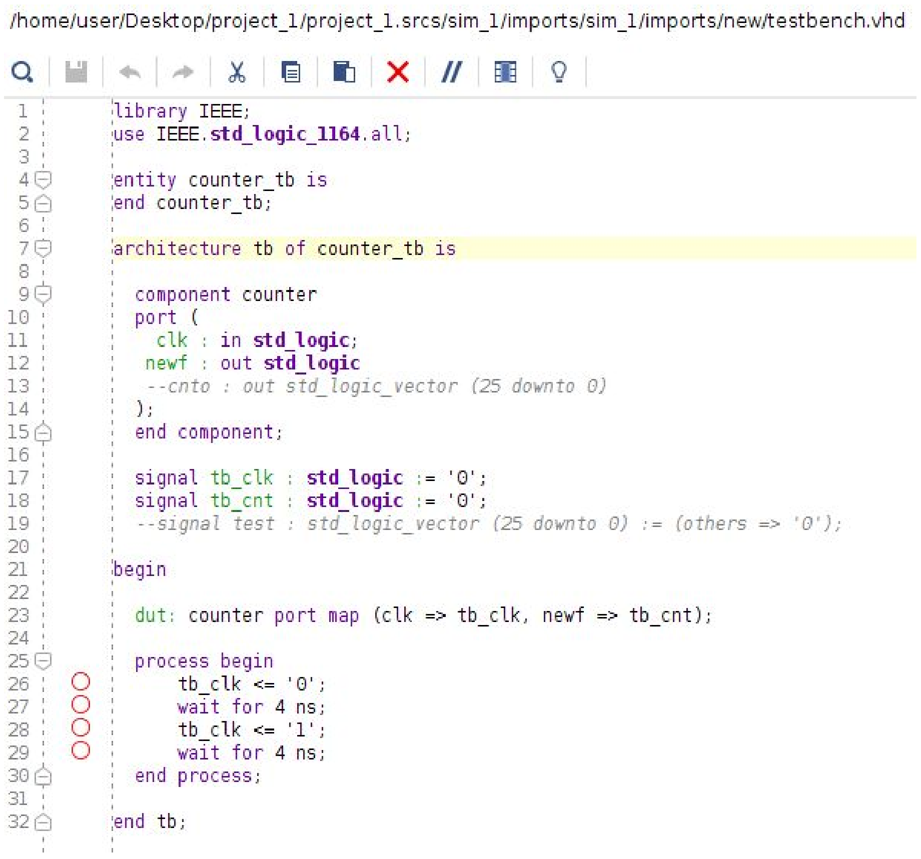
divider_top.vhd *

/home/user/Desktop/project_1/project_1.srcs/sources_1/imports/sources_1/imports/project_1.srcs/sources_1/new/divider_top.vhd

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity clock_div is
port (
  clk_in : in std_logic;
  div : out std_logic
);
end clock_div;

architecture cnt of clock_div is
  signal count : std_logic_vector (25 downto 0) := (others => '0');
begin
process(clk_in) begin
  if rising_edge(clk_in) then
    count <= std_logic_vector( unsigned(count) + 1);

    if ( count="11101110011010110010100000") then
      div <= '1';
      count <= (others => '0');

    else
      div <= '0';
    end if;

  end if;
end process;

end cnt;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity divider_top is
  Port ( clk : in std_logic;
         Q : out std_logic);
end divider_top;

architecture Behavioral of divider_top is

component clock_div is
port (
  clk_in : in std_logic;
  div : out std_logic
);
end component;

signal D, CE : std_logic := '0';

begin

di: clock_div
port map ( clk_in => clk,
           div => CE);

reg: process(clk)
begin
    if (rising_edge(clk)) then
        if(CE = '1') then
            D <= not D;
        end if;
    end if;
    end process;
    Q <= D;

end Behavioral;
```

Test

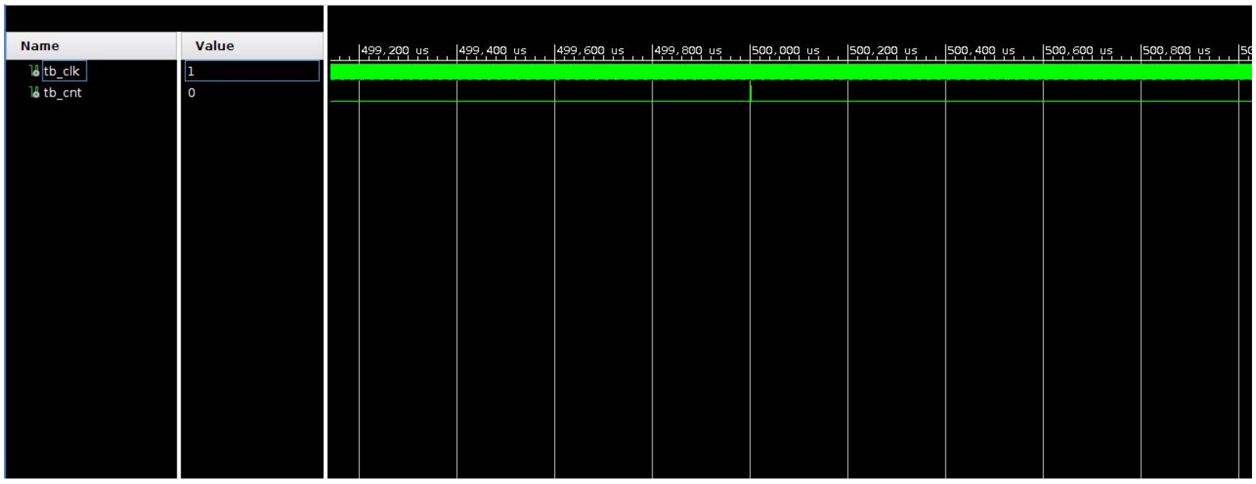
Test Bench



/home/user/Desktop/project_1/project_1.srcs/sim_1/imports/sim_1/imports/new/testbench.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity counter_tb is
end counter_tb;

architecture tb of counter_tb is

  component counter
  port (
     clk : in std_logic;
    newf : out std_logic
    --cnto : out std_logic_vector (25 downto 0)
  );
  end component;

  signal tb_clk : std_logic := '0';
  signal tb_cnt : std_logic := '0';
  --signal test : std_logic_vector (25 downto 0) := (others => '0');

begin

  dut: counter port map (clk => tb_clk, newf => tb_cnt);

  process begin
      tb_clk <= '0';
      wait for 4 ns;
      tb_clk <= '1';
      wait for 4 ns;
  end process;

end tb;
```

Simulation

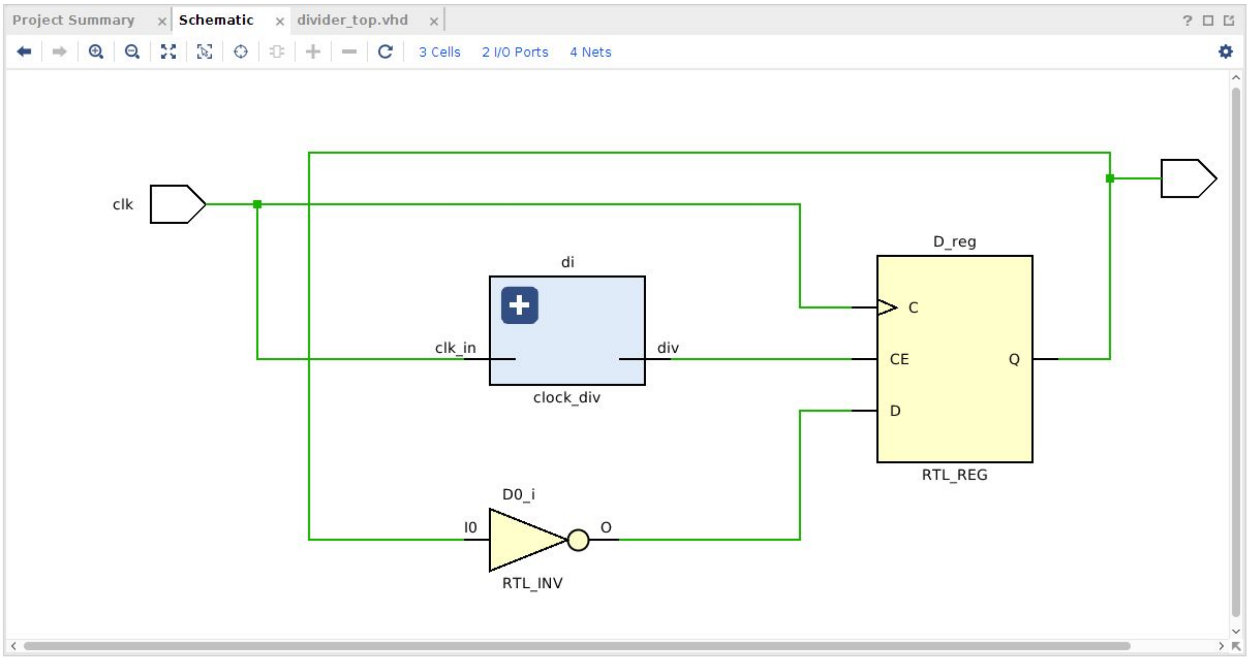


Name	Value
tb_clk	1
tb_cnt	0

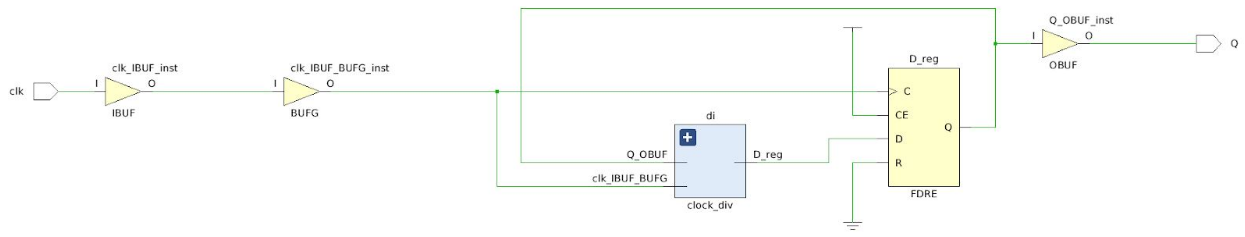
As you can see, the circuit outputs a pulse every 500 ms or 0.5 seconds.

Implementation

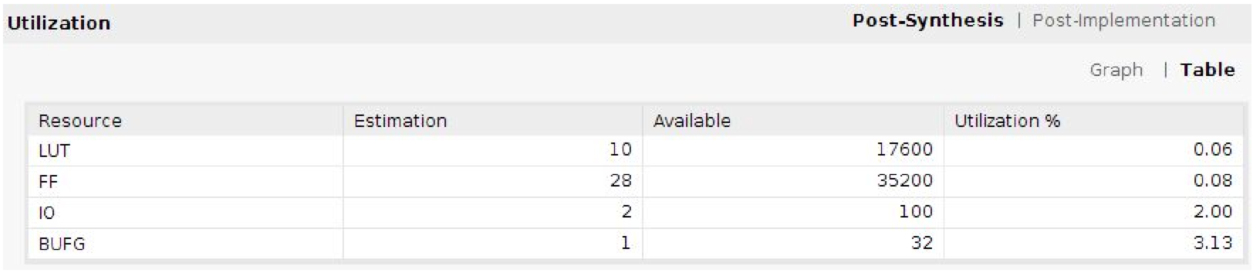
RTL Schematic



Synthesis Schematic



Post-Synthesis Utilization Table

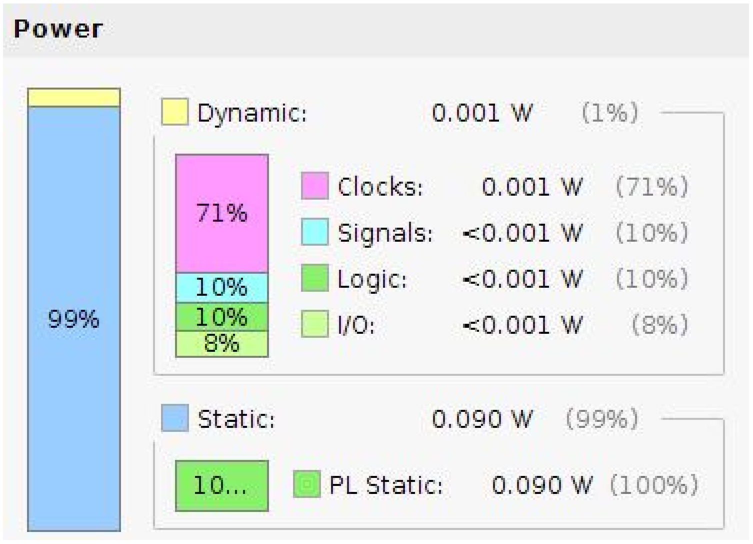


Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	10	17600	0.06
FF	28	35200	0.08
IO	2	100	2.00
BUFG	1	32	3.13

On-Chip Power Graph



Dynamic: 0.001 W (1%)

Clocks: 0.001 W (71%)

Signals: <0.001 W (10%)

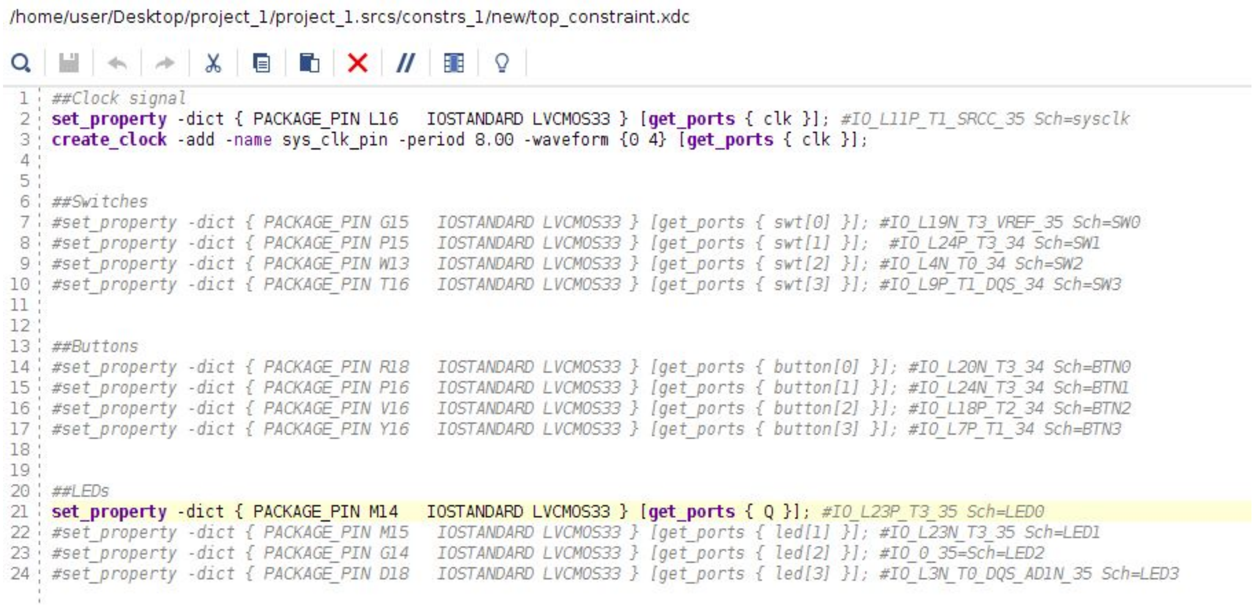
Logic: <0.001 W (10%)

I/O: <0.001 W (8%)

Static: 0.090 W (99%)

PL Static: 0.090 W (100%)

Constraint File



/home/user/Desktop/project_1/project_1.srcs/constrs_1/new/top_constraint.xdc

```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
#set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { swt[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
#set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { swt[1] }];  #IO_L24P_T3_34 Sch=SW1
#set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { swt[2] }]; #IO_L4N_T0_34 Sch=SW2
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { swt[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { button[0] }]; #IO_L20N_T3_34 Sch=BTN0
#set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { button[1] }]; #IO_L24N_T3_34 Sch=BTN1
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { button[2] }]; #IO_L18P_T2_34 Sch=BTN2
#set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { button[3] }]; #IO_L7P_T1_34 Sch=BTN3


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { Q }]; #IO_L23P_T3_35 Sch=LED0
#set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
#set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
#set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3
```

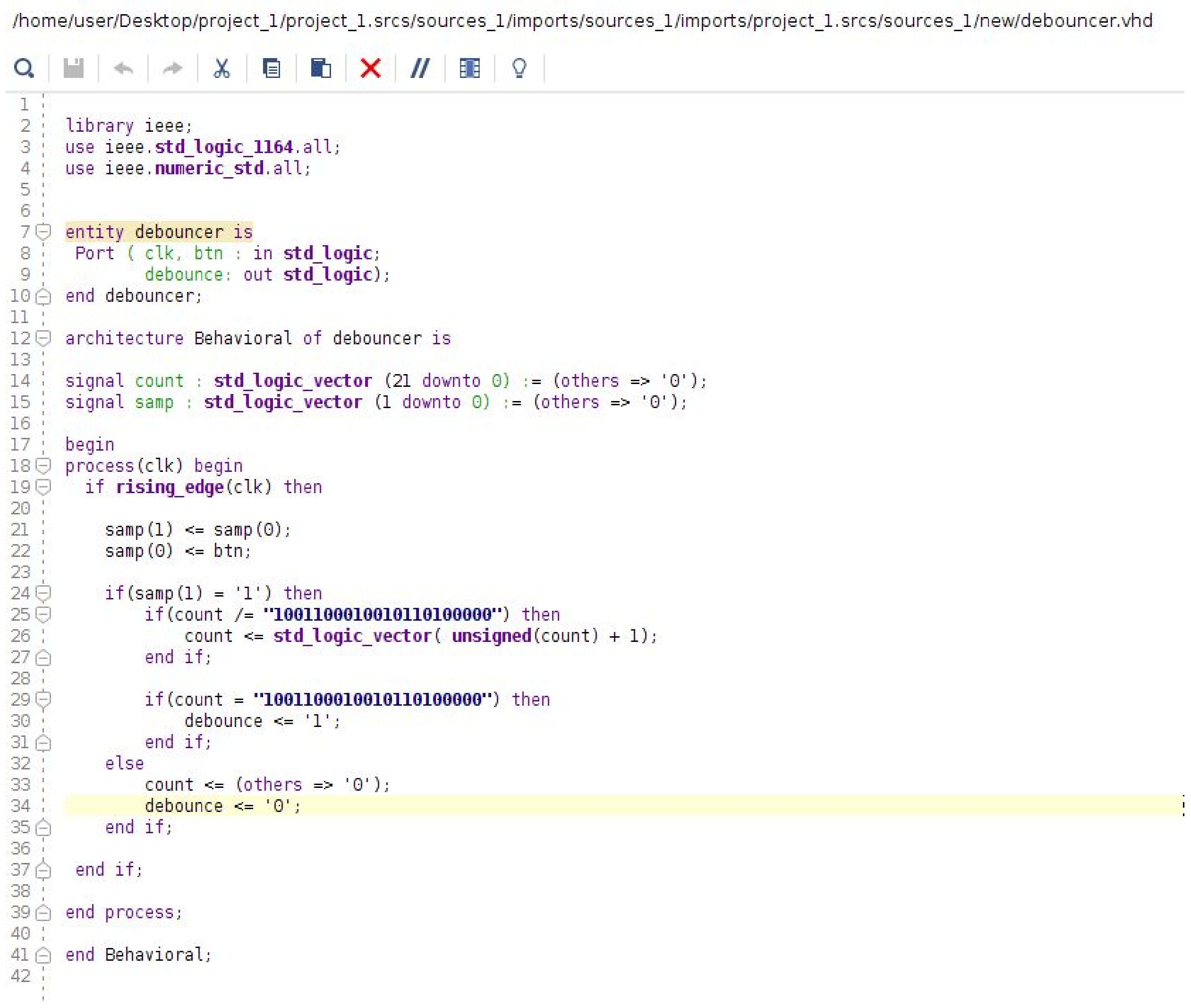
2. Bouncy Buttons

Theory

The second part of the lab presents another usage for the flexible counter. Buttons are mechanical devices that use a spring to pop back up after they are pushed. However, they do not perfectly rise back up. After being pushed, the spring jumps back up and oscillates before returning to its steady state. This presents a problem in digital circuit design. In digital circuits, this oscillation creates unwanted inputs as the springs bounces across the border of a logic high and a logic low. To fix this problem, the lab presents a debouncer circuit using a counter. This circuit works by incrementing a counter while the button input remains a logic ‘1’. If an input comes in a logic ‘0’, the counter is reset. When the counter reaches a certain value, it stop incrementing and outputs a ‘1’.

Design

Code:

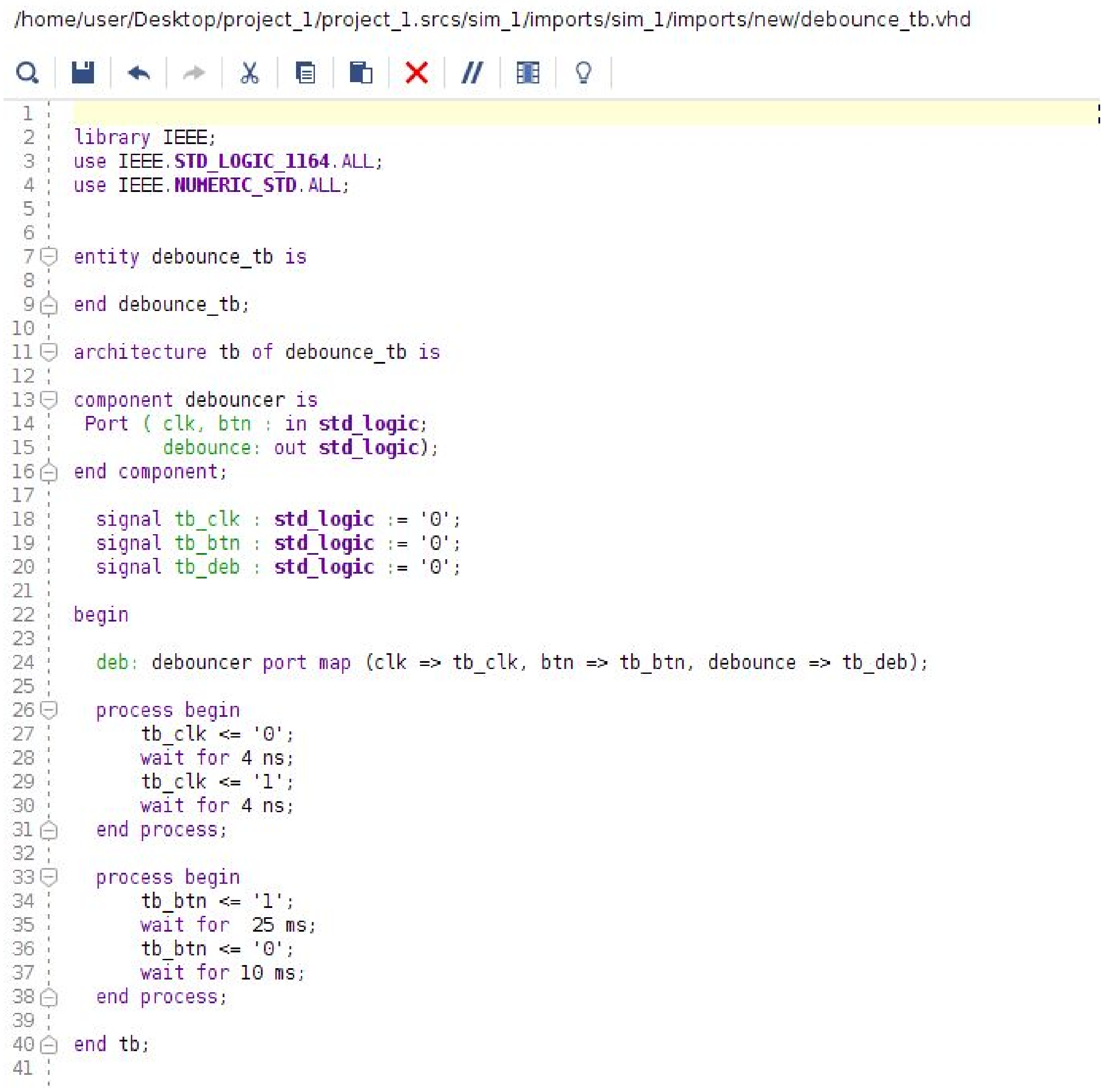


/home/user/Desktop/project_1/project_1.srcs/sources_1/imports/sources_1/imports/project_1.srcs/sources_1/new/debouncer.vhd

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity debouncer is
 Port ( clk, btn : in std_logic;
        debounce: out std_logic);
end debouncer;

architecture Behavioral of debouncer is

signal count : std_logic_vector (21 downto 0) := (others => '0');
signal samp : std_logic_vector (1 downto 0) := (others => '0');

begin
process(clk) begin
  if rising_edge(clk) then

    samp(1) <= samp(0);
    samp(0) <= btn;

    if(samp(1) = '1') then
        if(count /= "1001100010010110100000") then
            count <= std_logic_vector( unsigned(count) + 1);
        end if;

        if(count = "1001100010010110100000") then
            debounce <= '1';
        end if;
    else
        count <= (others => '0');
        debounce <= '0';
    end if;

 end if;

end process;

end Behavioral;

```

Test

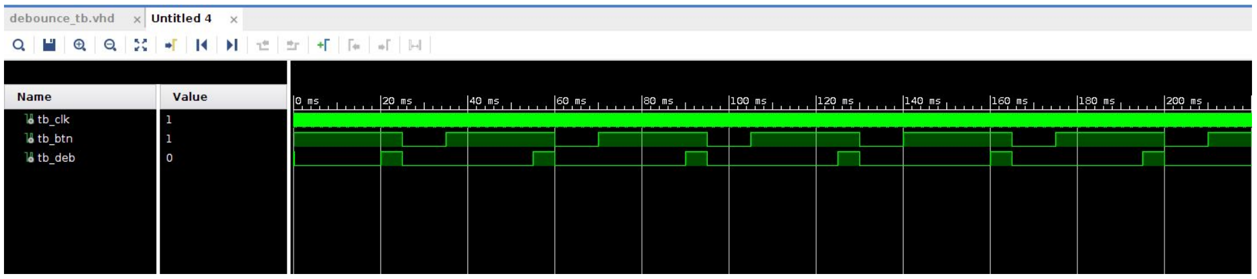
Test Bench



/home/user/Desktop/project_1/project_1.srcs/sim_1/imports/sim_1/imports/new/debounce_tb.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity debounce_tb is

end debounce_tb;

architecture tb of debounce_tb is

component debouncer is
 Port ( clk, btn : in std_logic;
        debounce: out std_logic);
end component;

  signal tb_clk : std_logic := '0';
  signal tb_btn : std_logic := '0';
  signal tb_deb : std_logic := '0';

begin

  deb: debouncer port map (clk => tb_clk, btn => tb_btn, debounce => tb_deb);

  process begin
      tb_clk <= '0';
      wait for 4 ns;
      tb_clk <= '1';
      wait for 4 ns;
  end process;

  process begin
      tb_btn <= '1';
      wait for  25 ms;
      tb_btn <= '0';
      wait for 10 ms;
  end process;

end tb;

```

Simulation



As seen in the test bench, when the button is pushed for 25ms, it is off for the first 20 ms while the counter is incrementing. When the counter reaches the desired value, the debouncer outputs a ‘1’ for the remaining 5ms that the button is held.

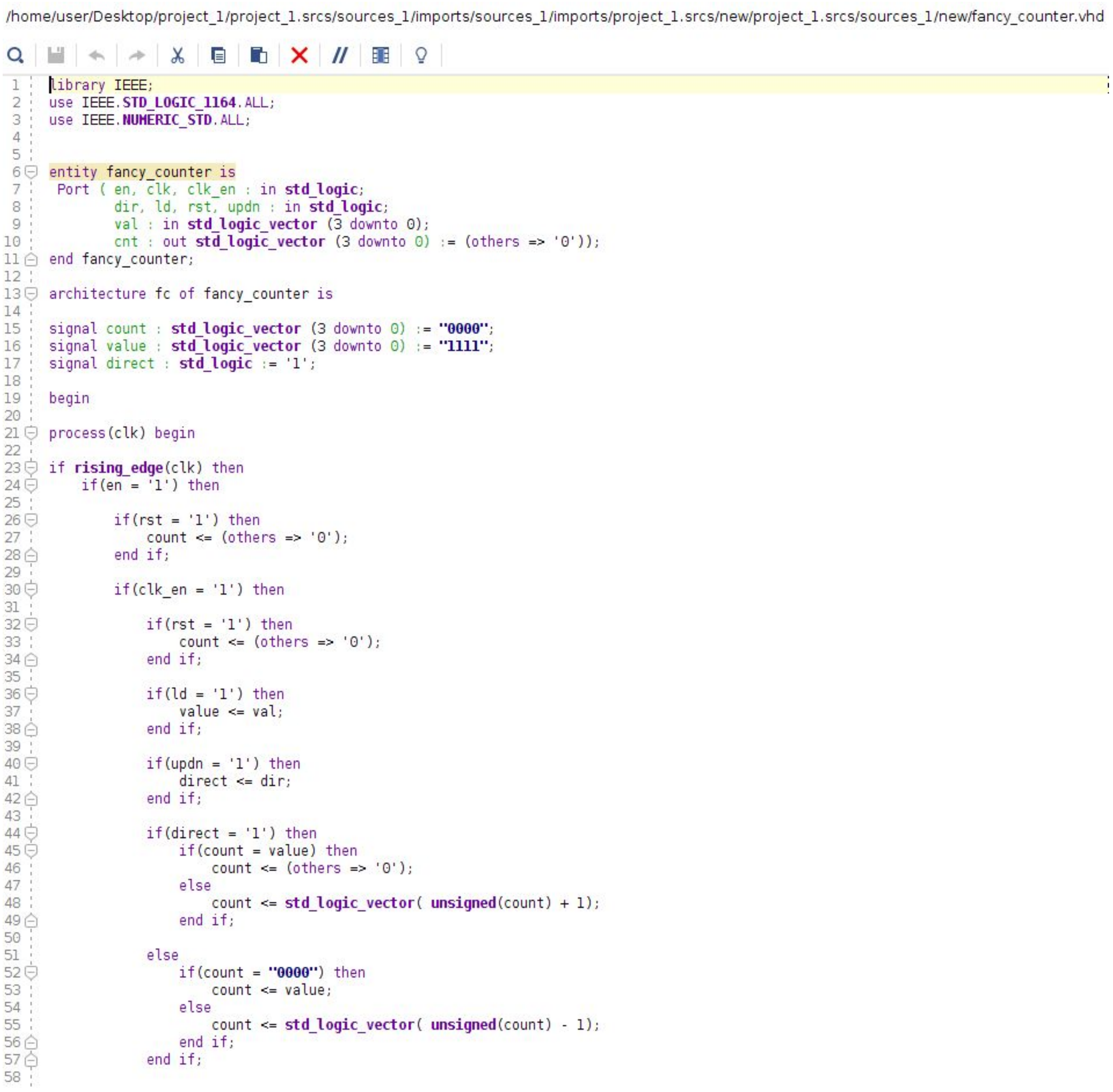
3. Actually Using a Counter to Count

Theory

This part of the lab is pretty self-explanatory. We use our basic counter to design a complete 4-bit counter. The 4-bit counter has two inputs to enable functionality in the device. It also has a reset input that sets the output count to 0. The 4-bit counter can count up or down depending on the signal going into the dir input. The dir input is only updated when the updn input is ‘1’. You can load the value being sent into the val input into the device by sending a logic ‘1’ to the ld input. The counter will count up to this value and then reset to 0. Or, if it is decrementing, it will count down to 0 before going back to the value.

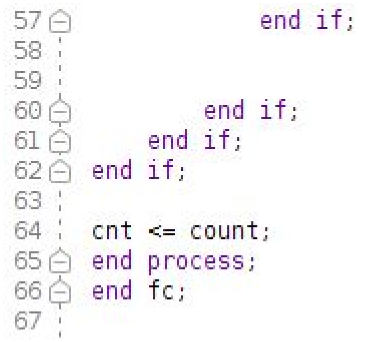
Design

Code



/home/user/Desktop/project_1/project_1.srcs/sources_1/imports/sources_1/imports/project_1.srcs/new/project_1.srcs/sources_1/new/fancy_counter.vhd

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity fancy_counter is
 Port ( en, clk, clk_en : in std_logic;
        dir, ld, rst, updn : in std_logic;
        val : in std_logic_vector (3 downto 0);
        cnt : out std_logic_vector (3 downto 0) := (others => '0'));
end fancy_counter;

architecture fc of fancy_counter is

signal count : std_logic_vector (3 downto 0) := "0000";
signal value : std_logic_vector (3 downto 0) := "1111";
signal direct : std_logic := '1';

begin

process(clk) begin

if rising_edge(clk) then
    if(en = '1') then

        if(rst = '1') then
            count <= (others => '0');
        end if;

        if(clk_en = '1') then

            if(rst = '1') then
                count <= (others => '0');
            end if;

            if(ld = '1') then
                value <= val;
            end if;

            if(updn = '1') then
                direct <= dir;
            end if;

            if(direct = '1') then
                if(count = value) then
                    count <= (others => '0');
                else
                    count <= std_logic_vector( unsigned(count) + 1);
                end if;

            else
                if(count = "0000") then
                    count <= value;
                else
                    count <= std_logic_vector( unsigned(count) - 1);
                end if;
            end if;

```



```vhdl
            end if;


        end if;
    end if;
end if;

cnt <= count;
end process;
end fc;

```

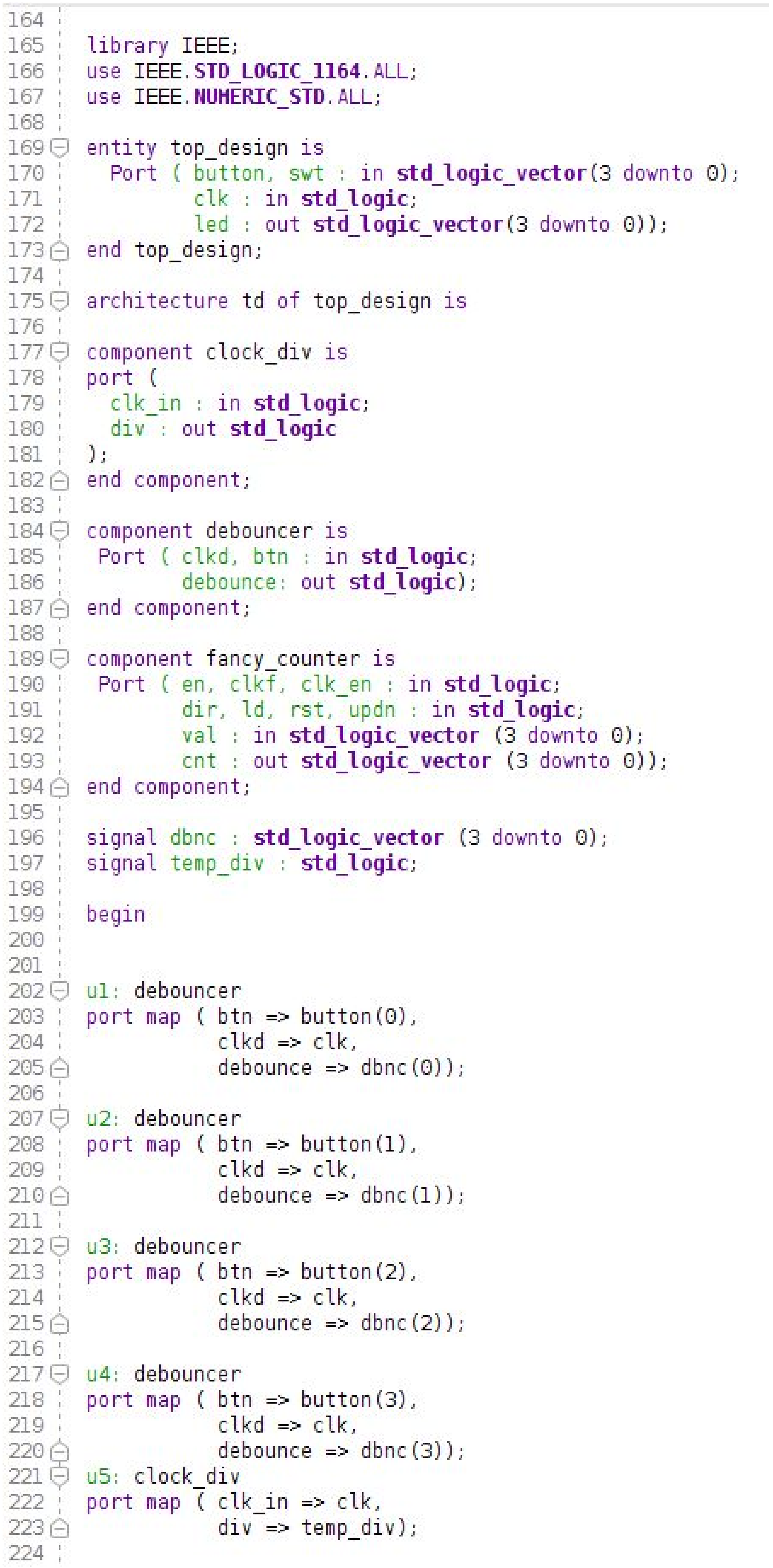
4. Bringing It All Together

Theory

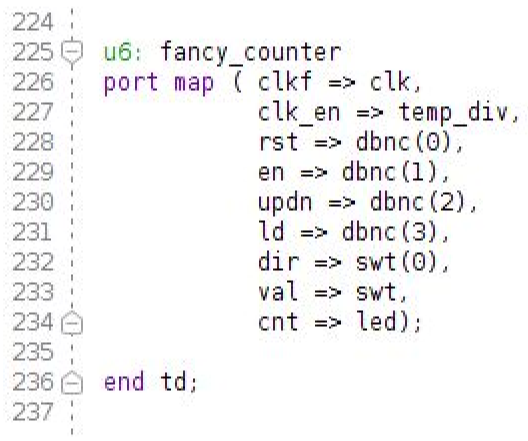
This part of the lab brings together all the parts designed in the previous sections of the lab. The final put together circuit is designed to be able to be implemented on a Zybo Board. The buttons on the board are fed into our debouncers to provide an accurate input signal. The output of these buttons are fed into the 4-bit counter. Switches on the board are used to control the val input on the 4-bit counter board and the clock divider is used to send a 2 Hz signal to one of the enabler inputs. Finally, the count output of the 4-bit counter is displayed using leds.

Design

Code



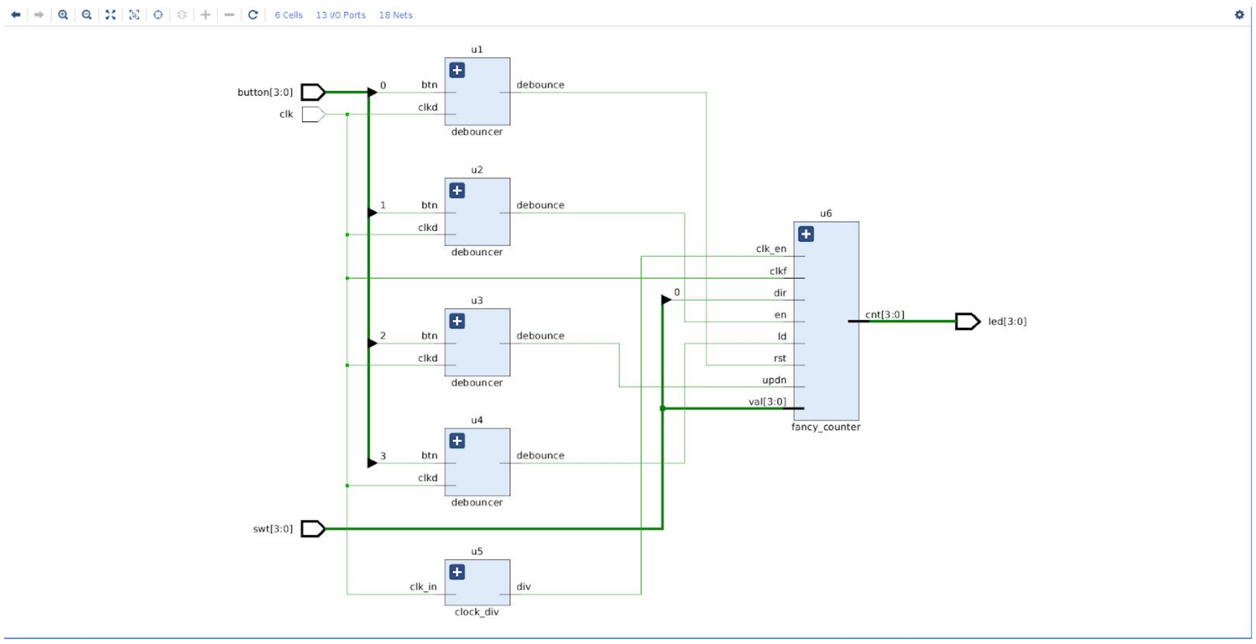
```vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity top_design is
  Port ( button, swt : in std_logic_vector(3 downto 0);
         clk : in std_logic;
         led : out std_logic_vector(3 downto 0));
end top_design;

architecture td of top_design is

component clock_div is
port (
  clk_in : in std_logic;
  div : out std_logic
);
end component;

component debouncer is
 Port ( clkd, btn : in std_logic;
        debounce: out std_logic);
end component;

component fancy_counter is
 Port ( en, clkf, clk_en : in std_logic;
        dir, ld, rst, updn : in std_logic;
        val : in std_logic_vector (3 downto 0);
        cnt : out std_logic_vector (3 downto 0));
end component;

signal dbnc : std_logic_vector (3 downto 0);
signal temp_div : std_logic;

begin


u1: debouncer
port map ( btn => button(0),
           clkd => clk,
           debounce => dbnc(0));

u2: debouncer
port map ( btn => button(1),
           clkd => clk,
           debounce => dbnc(1));

u3: debouncer
port map ( btn => button(2),
           clkd => clk,
           debounce => dbnc(2));

u4: debouncer
port map ( btn => button(3),
           clkd => clk,
           debounce => dbnc(3));
u5: clock_div
port map ( clk_in => clk,
           div => temp_div);

```



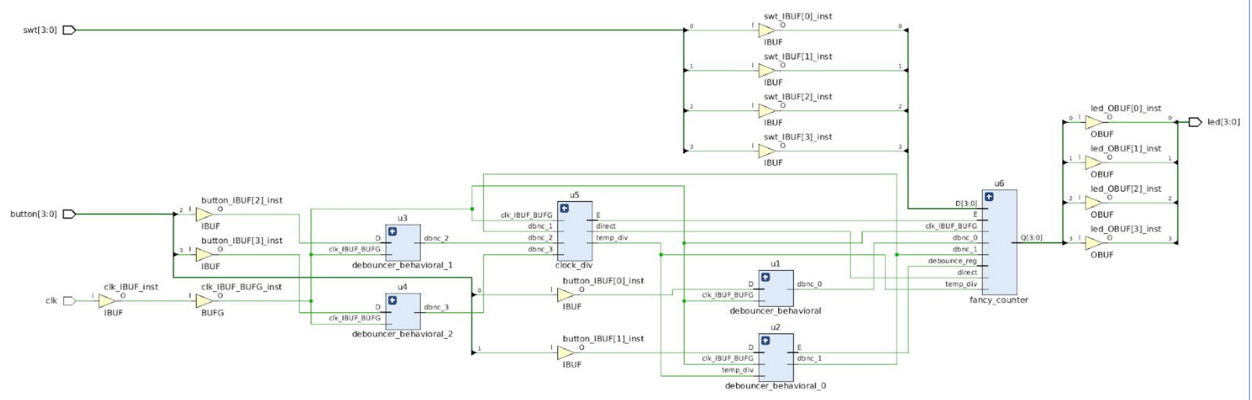
```
u6: fancy_counter
port map ( clkf => clk,
           clk_en => temp_div,
           rst => dbnc(0),
           en => dbnc(1),
           updn => dbnc(2),
           ld => dbnc(3),
           dir => swt(0),
           val => swt,
           cnt => led);

end td;
```

Implementation

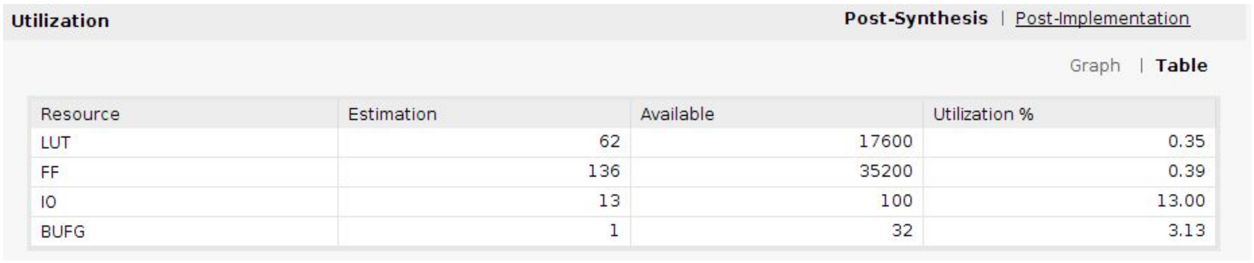
RTL Design



Synthesis Schematic



Post-Synthesis Utilization Table

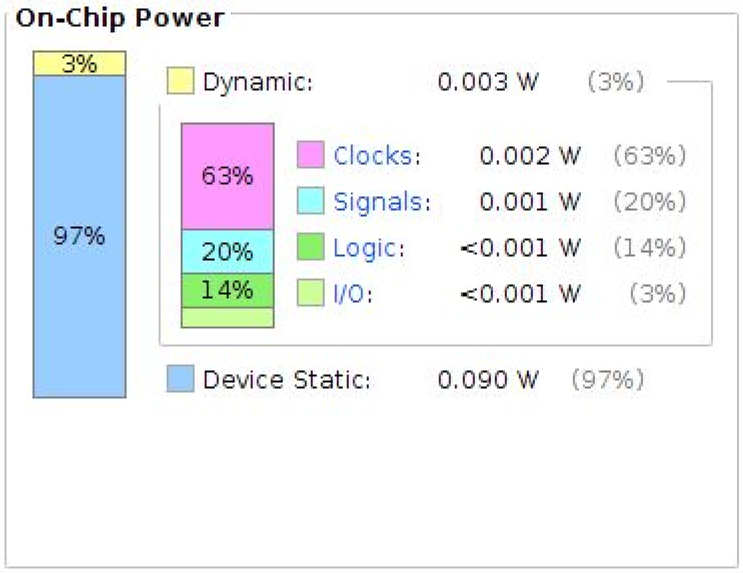


Utilization — **Post-Synthesis** | Post-Implementation

Graph | **Table**

Resource	Estimation	Available	Utilization %
LUT	62	17600	0.35
FF	136	35200	0.39
IO	13	100	13.00
BUFG	1	32	3.13

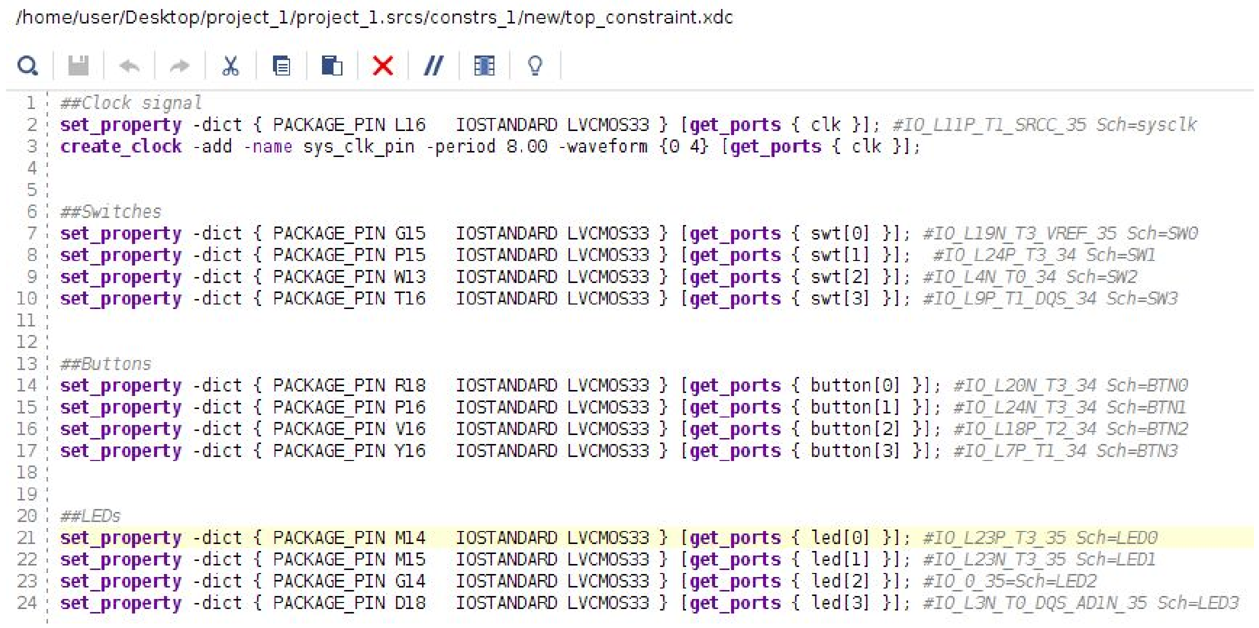
On-Chip Power



On-Chip Power

🟨	Dynamic:	0.003 W	(3%)
🟪	Clocks:	0.002 W	(63%)
🟦	Signals:	0.001 W	(20%)
🟩	Logic:	<0.001 W	(14%)
🟩	I/O:	<0.001 W	(3%)
🟦	Device Static:	0.090 W	(97%)

Constraint File



/home/user/Desktop/project_1/project_1.srcs/constrs_1/new/top_constraint.xdc

```
##Clock signal
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { clk }]; #IO_L11P_T1_SRCC_35 Sch=sysclk
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports { clk }];


##Switches
set_property -dict { PACKAGE_PIN G15   IOSTANDARD LVCMOS33 } [get_ports { swt[0] }]; #IO_L19N_T3_VREF_35 Sch=SW0
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { swt[1] }];  #IO_L24P_T3_34 Sch=SW1
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 } [get_ports { swt[2] }]; #IO_L4N_T0_34 Sch=SW2
set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { swt[3] }]; #IO_L9P_T1_DQS_34 Sch=SW3


##Buttons
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { button[0] }]; #IO_L20N_T3_34 Sch=BTN0
set_property -dict { PACKAGE_PIN P16   IOSTANDARD LVCMOS33 } [get_ports { button[1] }]; #IO_L24N_T3_34 Sch=BTN1
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { button[2] }]; #IO_L18P_T2_34 Sch=BTN2
set_property -dict { PACKAGE_PIN Y16   IOSTANDARD LVCMOS33 } [get_ports { button[3] }]; #IO_L7P_T1_34 Sch=BTN3


##LEDs
set_property -dict { PACKAGE_PIN M14   IOSTANDARD LVCMOS33 } [get_ports { led[0] }]; #IO_L23P_T3_35 Sch=LED0
set_property -dict { PACKAGE_PIN M15   IOSTANDARD LVCMOS33 } [get_ports { led[1] }]; #IO_L23N_T3_35 Sch=LED1
set_property -dict { PACKAGE_PIN G14   IOSTANDARD LVCMOS33 } [get_ports { led[2] }]; #IO_0_35=Sch=LED2
set_property -dict { PACKAGE_PIN D18   IOSTANDARD LVCMOS33 } [get_ports { led[3] }]; #IO_L3N_T0_DQS_AD1N_35 Sch=LED3
```

Discussion

Questions

1.1. How much do we need to divide our input by to get from 125 MHz to 2 Hz?

    125/x = 2 → x = 62.5MHz

1.2 How many bits are required to store a counter that can count up to the value obtained in Q1.1?

    When converting 62.5MHz you get a binary number that takes 26 bits.

Question 2.1: What is the value of the button when it is pressed for the Zybo?

    When a button is pressed the Zybo inputs a logic ‘1’

Question 2.3: If we want our debounce time to be 20 ms, and our system clock is 125 MHz, how many ticks do we need a steady ‘1’ to be read for it to count as a ‘1’ ?

    125 MHz is in ticks/second. So in order to get the number of ticks we multiply 125 MHz with 20 ms. This gives us 2500000 ticks.

Question 2.4: How many bits are required for a counter that can go that high?

    The counter needs 22 bits.

Observations

Although a little confusing at times, the lab did a pretty good job of introducing me to designing digital circuits. Through trial and error I have learned about many mistakes that one could make when coding in VHDL and how to avoid them. I've also gotten more comfortable with using port mapping in VHDL. Before this lab I thought I had to copy my entire component code into the overall design file. Near the end I realized that you can map to other source files without having to migrate the whole code. Additionally, the lab helped me get accustomed with counters and how to use them. I now have a glimpse into how counters can be used and will keep them in mind when designing new digital circuits.

Questions

After finishing the lab, I feel that I have a pretty good understanding of how VHDL works and is implemented. One thing I still do not completely understand is errors in code. Often times, I have a hard time understanding where the error is coming from and when the issue is. For example, when simulating my fancy_counter, my count signal was displaying a value of X. This means that it was getting conflicting values sent to it. I managed to fix it but I am unsure what I did. I believe it has something to do with my multiple processes inside my test bench but I am unsure of the exact issue. Following up on that, I am a little unsure about how test benches and multiple simultaneous processes work.